Available online at www.sciencedirect.com

SCIENCE DIRECT

Electronic Notes in Theoretical Computer Science

Electronic Notes in Theoretical Computer Science 128 (2005) 3–19

www.elsevier.com/locate/entcs

A Generic Cost Model for Concurrent and Data-parallel Meta-computing

Armelle Merlin [1]

LIFO, Université d'Orléans - CNRS
Orléans, France

Gaétan Hains[2]

LIFO, Université d'Orléans - CNRS
Orléans, France

Abstract

The CCS (Calculus of Communicating System) process algebra is a well-known formal model of synchronization and communication, useful for the analysis of safety and liveness in protocols or distributed programs, and in more recent works their security properties. BSP (Bulk-synchronous parallelism) is an algorithm- and programming model of data-parallel computation. It is useful for the design, analysis and programming of scalable parallel algorithms.
Many current evolutions require the integration of distributed- and parallel programming: grid systems for sharing resources across the Internet, secure and reliable global access to parallel computer systems, geographic distribution of confidential data on randomly accessible systems, etc. Such software services must provide guarantees of safety, liveness, and security together with scalable and reliable performance. Formal models are therefore needed to combine parallel performance and concurrent behavior. With this goal in mind, we propose here an integration of BSP with CCS semantics, generalize its cost (performance) model and sketch its application to scheduling problems in meta-computing.

Keywords: Concurrency, cost model, data-parallelism, CCS, BSP

[1] Email: Armelle.Merlin@lifo.univ-orleans.fr
[2] Email: ghains@lifo.univ-orleans.fr

doi:10.1016/j.entcs.2005.04.001

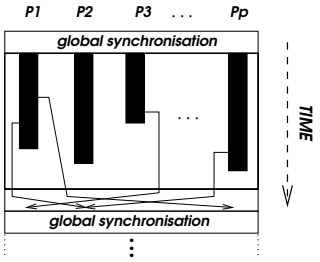
1 Introduction

We will first present here preliminaries on BSP and CCS. In the first section we will propose a CCS extension to express BSP-like processes. The second section will present the formal usual tools for a process algebra. We then describe a formal cost model for this extended algebra in the standard graph theoretic manner: to associate "customized" semi-ring elements to sets of paths in the transition systems of CCS processes. We finally outline its application to scheduling problems in meta-computing, and some future developments. Full technical developments are available in a technical report [8].

1.1 Preliminary on the BSP model

The BSP execution model [13] represents a parallel computation on p identical processors as an alternating sequence of computation *super-steps* (p asynchronous computations) and communications supersteps (data exchanges between processors with global synchronization). The following figure represents a BSP superstep. The BSP *cost* model estimates execution times as multiples of the time for a sequential operation by a simple formula based on p, on the sequential speed of processors, and the network bandwidth g and latency L. The execution time of a computation superstep followed by a synchronization superstep is estimated by

$$Time = \max_{0 \le i < p} w_i + \max_{0 \le i < p} h_i * g + L$$



where $h_i = \max(h_{i+}, h_{i-})$ and h_{i+} (resp. h_{i-}) is the total size of the messages sent (resp. received) by the processor i during the communication superstep. Here w_i is the sequential speed of the processor i during the computation superstep.

BSP is thus a simplified and portable parallel architecture model, useful for algorithm design, scalability analysis and programming. An ML-like BSP programming language, called BS-λ [7], has been formalized by a BSP extension of the λ-calculus. That model is appropriate for pure data-parallelism but lacks expressive power to describe meta-computing systems, which leads to the work described further.

1.2 Preliminary on CCS

The following subsection is a summary of [10] and [9].

1.2.1 Definitions

All process expressions are in the set $\mathcal{E}$. $\mathcal{A}$ is an infinite set of *names* or *input labels*, and $\overline{\mathcal{A}}$ is the set of *co-names* or *output labels*. $\mathcal{A}$ and $\bar{\mathcal{A}}$ are disjoint and are in bijection via $(\bar{\ })$. $\mathcal{L} = \mathcal{A} \cup \overline{\mathcal{A}}$ is the set of *labels*. $\alpha \in \mathcal{A}$ and $\bar{\alpha} \in \overline{\mathcal{A}}$ are said to be *inverses*. Labels are also called *actions* or *events*.

Here is the syntax of a process (or an *agent expression*) where $\alpha \in \mathcal{L}$ and X is any process variable:

- $P ::= 0 \mid \alpha.P \mid P + P \mid P\backslash\alpha \mid \texttt{let}\ X = P\ \texttt{in}\ P \mid \texttt{rec}\ X : P \mid (P|P) \mid X$
- $\texttt{Act} ::= \alpha \mid \tau$ where $\alpha \in \mathcal{L}$. *Remark: τ has no inverse.*

1.2.2 Rules

The transition semantics of the local terms is given in the following figure. The rule ACT relates to the prefix operator, SUM relates to summation, RES to restriction, LET to relabeling, REC to recursion, ASYNCL, ASYNCR and ASYNCR are the parallel composition's left-asynchronous, right-asynchronous or synchronous transition rules.

$$\texttt{ACT}\quad \frac{}{\alpha.P \xrightarrow{\alpha} P}$$

$$\texttt{SUM}_0\quad \frac{P_i \xrightarrow{\alpha} P_i'}{P_i + P_j \xrightarrow{\alpha} P_i'} \qquad \texttt{REC}\quad \frac{P[X \leftarrow \mathrm{rec}X : P] \xrightarrow{\alpha} P'}{\texttt{rec}X : P \xrightarrow{\alpha} P'}$$

$$\texttt{SUM}_1\quad \frac{P_j \xrightarrow{\alpha} P_j'}{P_i + P_j \xrightarrow{\alpha} P_j'} \qquad \texttt{ASYNCL}\quad \frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q}$$

$$\texttt{RES}\quad \frac{P \xrightarrow{\alpha} P'}{P\backslash\beta \xrightarrow{\alpha} P'\backslash\beta}(\alpha \notin \{\beta, \bar{\beta}\}) \qquad \texttt{PAR}\quad \frac{P \xrightarrow{\alpha} P'\ \ Q \xrightarrow{\bar{\alpha}} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$$

$$\texttt{LET}\quad \frac{Q[X \leftarrow P] \xrightarrow{\alpha} R}{\texttt{let}\ X = P\ \texttt{in}\ Q \xrightarrow{\alpha} R} \qquad \texttt{ASYNCR}\quad \frac{Q \xrightarrow{\alpha} Q'}{P|Q \xrightarrow{\alpha} P|Q'}$$

Remark 1.1 The notation LET IN is used to share terms in sub-terms (without recursion), and the notation $P[X \leftarrow Q]$ is the mathematical substitution ($[X \leftarrow Q]$ is a meta-operation and not part of the process algebra syntax).

1.2.3 Strong bisimulation

Milner proposes a binary relation on $\mathcal{E}$, the *s*trong bisimulation denoted by $\sim$. He exposes a way to eliminate the parallel composition $|$, the restriction and the relabeling to obtain a bisimilar *s*erial term thanks to an *e*xpansion law.

If a serial term contains no recursion then it is a *finite* serial term. Milner proposes an axiom system $\mathcal{A}_1$ on finite serial agents with $+$ commutative, associative, idempotent and 0 as unit of $+$.

Proposition 1.2 *Let P and Q be finite serial terms. $P \sim Q$ iff $\mathcal{A}_1 \vdash P = Q$*

2 A data-parallel process algebra

In this section, we describe the syntax and semantics of BSPA (BSP Process Algebra). As in BSλ, the process algebra is divided in a local (resp. a global level) representing individual processors from parallel systems (resp. combinations of parallel systems or the isolated "console" processor implicit in CCS semantics).

2.1 Local processes

The local level is CCS with slight changes. The actions on this level can be done locally by any processor of a parallel system.

2.1.1 Definitions

We introduce a local label `put` which expresses the specific BSP exchange. This label is associated with a set of substitutions σ which correspond to the data to be exchanged. λ is a subset of σ and is also a set of substitutions. This construction corresponds to the encapsulation of data depending on the destination of these data to be sent.

- $\texttt{Act} ::= \alpha \mid \texttt{put}[\sigma] \mid \tau$ where $\alpha \in \mathcal{L}$. *Remark: τ and* `put` *have no inverse.*
- $\sigma ::= [i \leftarrow \lambda]^*$ the syntactic category of substitutions to express the exchange due to a `put`. i is the number of the processor where the λ substitutions are sent. This category can be empty.
- $\lambda ::= [X \leftarrow P]^*$ the syntactic category Subst of substitutions that are to be sent. This category can also be empty.

2.1.2 Local rules

The transition semantics of the local terms is the same as the CCS one. As the $\texttt{put}[\sigma]$ action appears only on the global level rules.

2.2 Global processes

Next, we present the global level. The definitions, syntax and semantics are very similar to those of the local level, except for the synchronization barrier.

For this section, we will always refer to a vector of fixed size which represents a parallel machine of fixed size p (BSλ-like [7]).

2.2.1 Definitions

$\mathcal{E}_v$ is the set of global (process) expressions. $\mathcal{E}_{v,FIN}$ is the set of global expressions without `rec`. We define

- these following sets: input labels $\mathcal{A}_V = \mathcal{A} \cup \{\langle \alpha @ i \rangle_p \mid i \in 0, .., p-1, \alpha \in \mathcal{A}\}$, output labels $\bar{\mathcal{A}}_V = \bar{\mathcal{A}} \cup \{\langle \bar{\alpha} @ i \rangle_p \mid i \in 0, .., p-1, \bar{\alpha} \in \bar{\mathcal{A}}\}$ and $\mathcal{L}_V = \mathcal{A}_V \cup \bar{\mathcal{A}}_V$.
- the processor position numbers $\mathtt{i} ::= 0 \mid 1 \mid \ \ldots \ \mid p-1$
- the terms $V ::= 0_v \mid \alpha.V \mid V + V \mid V \backslash \alpha \mid \mathtt{let}\ X = V\ \mathtt{in}\ V \mid \mathtt{rec}\ X : V \mid (V|V) \mid \langle P, .., P \rangle_p \mid X$ where $\alpha \in \mathcal{L}_V$.
  For every $p \geq 1$ there are global process vectors $\langle P, .., P \rangle_p$, representing a group of local processes engaged in a collective data-parallel computation, such as an MPI *communicator*. The size is left unspecified when not relevant. The p processes are assumed to be allocated to p distinct processors.
- the actions $\mathtt{Act}_V ::= \alpha \mid T_{\sigma,\pi} \mid \langle \tau @ ij \rangle_p \mid \tau_{Net}$ where $\alpha \in \mathcal{L}_V$
  · $\langle \alpha @ i \rangle_p$ is the asynchronous occurrence of action α on processor i within a vector of size p. $\langle \alpha @ i \rangle_p$ can synchronize with the action $\bar{\alpha}$ or $\langle \bar{\alpha} @ j \rangle_p$ with $i \neq j$.
  · $T_{\sigma,\pi}$ is an abbreviation of $T_{\forall i \forall j \sigma_i \pi_j}$. It is analogous to τ for global synchronization. σ_i is the set of substitutions send by the processor i, and π_j is the set of substitutions received by the processor j.
  · $\langle \tau @ ij \rangle_p$ is the τ synchronization between two different ($i \neq j$) processes inside the parallel vector.
  · τ_{Net} is the synchronization between processors over the external network, i.e. between two parallel vectors.

2.2.2 Global rules

The transition semantics of global processes is defined by the set of rules in the following table.

$$\mathtt{ACT}\quad \frac{}{\alpha.V \xrightarrow{\alpha} V}$$

$$\mathtt{SUM_0}\quad \frac{V_i \xrightarrow{\alpha} V_i'}{V_i + V_j \xrightarrow{\alpha} V_i'} \qquad \mathtt{LET}\quad \frac{W[X \leftarrow V] \xrightarrow{\alpha} W'}{\mathtt{let}\ X = V\ \mathtt{in}\ W \xrightarrow{\alpha} W'}$$

$$\mathtt{SUM_1}\quad \frac{V_j \xrightarrow{\alpha} V_j'}{V_i + V_j \xrightarrow{\alpha} V_j'} \qquad \mathtt{REC}\quad \frac{V[X \leftarrow \mathrm{rec}X : V] \xrightarrow{\alpha} V'}{\mathtt{rec}\ X : V \xrightarrow{\alpha} V'}$$

$$\mathtt{RES}\quad \frac{V \xrightarrow{\alpha} V'}{V \backslash \beta \xrightarrow{\alpha} V' \backslash \beta} (\alpha \neq \beta \neq \bar{\beta}) \qquad \mathtt{ASYNCLV}\quad \frac{V_0 \xrightarrow{\alpha} V_0'}{V_0 | V_1 \xrightarrow{\alpha} V_0' | V_1}$$

$$\texttt{ASYNCRV}\quad \frac{V_1 \xrightarrow{\alpha} V_1'}{V_0|V_1 \xrightarrow{\alpha} V_0|V_1'} \qquad \texttt{PARV}\quad \frac{V_0 \xrightarrow{\alpha} V_0' \;\; V_1 \xrightarrow{\bar{\alpha}} V_1'}{V_0|V_1 \xrightarrow{\tau_{Net}} V_0'|V_1'}$$

$$\texttt{ASYNC}\quad \frac{P_i \xrightarrow{\alpha} P_i'}{\langle P_0,\ldots,P_i,\ldots,P_{p-1}\rangle_p \xrightarrow{\langle \alpha @ i\rangle_p} \langle P_0,\ldots,P_i',\ldots,P_{p-1}\rangle_p}\ \alpha \in \mathcal{L}\cup\{\tau\}$$

$$\texttt{DUOCHRONE}\quad \frac{P_i \xrightarrow{\alpha} P_i' \;\; P_j \xrightarrow{\bar{\alpha}} P_j' \;\; i\neq j}{\langle P_0,\ldots,P_i,\ldots,P_j\ldots,P_{p-1}\rangle_p \xrightarrow{\langle \tau @ ij\rangle_p} \langle P_0,\ldots,P_i',\ldots,P_j'\ldots,P_{p-1}\rangle_p}$$

$$\texttt{ISOCHRONE}\quad \frac{P_i \xrightarrow{\texttt{put}[\sigma_i]} P_i' \;\; \forall i\in\{0,\ldots,p-1\}}{\langle P_0,\ldots,P_{p-1}\rangle_p \xrightarrow{T_{\sigma,\pi}} \langle \pi_0(P_0'),\ldots,\pi_{p-1}(P_{p-1}')\rangle_p}\ \pi_i = \sigma_{(p-1)i}\circ\ldots\circ\sigma_{0i}$$

The rules ACT, $\texttt{SUM}_0$, $\texttt{SUM}_1$, RES, LET, REC, ASYNCLV, ASYNCRV and PARV are the usual ones. ASYNC is an emission (or reception), local at i inside the parallel vector. DUOCHRONE is an exchange between two processes inside the parallel vector, noting that $\alpha \neq \texttt{put}$. The rule ISOCHRONE is the semantics of the barrier. As presented in Definition 2.2.1 there is a constructor $\langle\ldots\rangle$ for each parallel machine. Processor i realizes the event $\texttt{put}[\sigma_i]$, σ_i is the set of the substitutions to be sent from P_i, and π_j is the set of substitutions received by P_j. To cope with possible concurrent substitution into the same variable, we define priority "to the left". The cost in time, a key ingredient of the BSP theory, will be defined in Section 4.2 from this measure of communication volume.

2.3 Examples

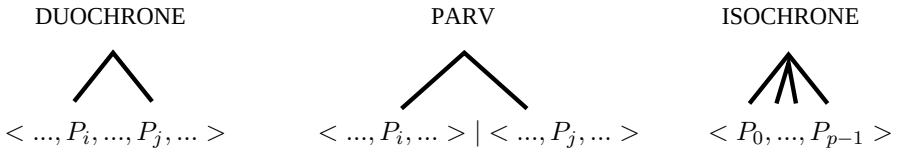


Fig. 1. Main cases of synchronization

- DUOCHRONE
  Here is an example of a four-processor parallel system. Processors 0 and 2 synchronize through rule DUOCHRONE.

$$\frac{\langle a,b,\bar{a},c\rangle_4 \xrightarrow{\langle a@0\rangle_2} \langle a,b,0,c\rangle_4 \;\; \langle a,b,\bar{a},c\rangle_4 \xrightarrow{\langle \bar{a}@2\rangle_4} \langle a,b,0,c\rangle_4}{\langle a,b,\bar{a},c\rangle_4 \xrightarrow{\langle \tau@02\rangle_4} \langle 0,b,0,c\rangle_4}$$

- ASYNC+PARV
  Two different parallel systems, represented by terms with root $\langle,,\rangle_p$, can synchronize as in CCS. If two synchronizations of this type are possible, they can not be simultaneous. This represents the CCS semantics we keep

as part of the theory, and means that the external network serializes communications.

$$\begin{array}{ccc} \langle a,b,c\rangle_3|\langle \bar{b},\bar{a}\rangle_2 & \stackrel{\tau_{Net}}{\rightarrow} & \langle 0,b,c\rangle_3|\langle \bar{b},0\rangle_2 \\ \downarrow \tau_{Net} & & \downarrow \tau_{Net} \\ \langle a,0,c\rangle_3|\langle 0,\bar{a}\rangle_2 & \stackrel{\tau_{Net}}{\rightarrow} & \langle 0,0,c\rangle_3|\langle 0,0\rangle_2 \end{array}$$

- ISOCHRONE with $p = 4$

  Figure 2 shows a communication superstep (and hence a synchronization barrier) between four processors.

$$\langle \mathtt{put}\begin{bmatrix} 1 \leftarrow [X \leftarrow \alpha] \\ 2 \leftarrow [X \leftarrow \beta] \end{bmatrix}.(X+Y), \mathtt{put}\begin{bmatrix} 0 \leftarrow [Y \leftarrow \gamma, X \leftarrow \delta] \\ 3 \leftarrow [X \leftarrow \alpha] \end{bmatrix}.X, \mathtt{put}\begin{bmatrix} 0 \leftarrow [X \leftarrow \alpha] \\ 1 \leftarrow [X \leftarrow \gamma] \\ 3 \leftarrow [Y \leftarrow \beta] \end{bmatrix}.X, \mathtt{put}[].(X+Y)\rangle_4$$

$$\downarrow T_{\sigma,\pi}$$

$$\langle \delta+\gamma, \alpha, \beta, \alpha+\beta\rangle_4$$

Fig. 2. Example: ISOCHRONE

3 Strong bisimulation

We extend the Milner's definition of bisimulation to the global level. This will allow us to validate transformation rules to reduce our process expressions to specific forms (explicit descriptions of the transition system) to define costs for them.

3.1 Properties of strong bisimulation for global terms

In this section we will assume that terms are closed in order to reduce them to serial forms. Our parallel system constructor reduces to CCS parallel composition in the absence of barrier synchronizations (modulo processor indices). This leads to Lemma 3.1

Lemma 3.1 $\langle P,Q\rangle_p \sim P|Q$ *if* $\mathtt{put} \notin P$ *and* $\mathtt{put} \notin Q$.

Lemma 3.2 $\langle P_0,\ldots,P_{p-1}\rangle_p \sim \langle P_{\sigma(0)},\ldots,P_{\sigma(p-1)}\rangle_p$ *where* σ *is any permutation.*

Proposition 3.3 *Sum* $+$ *for global terms is associative, commutative, idempotent and has* 0_V *as unit. Parallel composition* $|$ *for global terms is associative, commutative and has* 0_V *as unit.*

Proposition 3.4 *The following rules are used to eliminate the restriction* $\backslash$*:*

- $V\backslash\alpha \sim V$ *if* $\alpha \notin \mathcal{L}(V)$
- $V\backslash\alpha\backslash\alpha \sim V\backslash\alpha$
- $V\backslash\alpha\backslash\beta \sim V\backslash\beta\backslash\alpha$
- $(W|V)\backslash\alpha \sim W\backslash\alpha|V\backslash\alpha$ *if* $\alpha \notin \mathcal{L}(V) \cup \mathcal{L}(W)$.

Proposition 3.5 *Expansion law for data-parallel vectors*
Let $V \equiv \langle P_0, \ldots, P_{p-1}\rangle_p\backslash\beta$ *then*

$$\begin{aligned} V \sim & \sum \langle \alpha@i\rangle_p.\{\langle \ldots, P_{i-1}, P'_i, P_{i+1}, \ldots\rangle_p\backslash\beta : \exists i P_i \xrightarrow{\alpha} P'_i, \alpha \neq \beta, \alpha \neq \texttt{put}\} \\ & + \sum \langle \tau@ij\rangle_p.\{\langle \ldots, P'_i, \ldots, P'_j, \ldots\rangle_p\backslash\beta : P_i \xrightarrow{\alpha} P'_i, P_j \xrightarrow{\bar{\alpha}} P'_j, i \neq j\} \\ & + \sum T_{\sigma,\pi}.\{\langle P'_0, \ldots, P'_{p-1}\rangle_p\backslash\beta : \forall i P_i \xrightarrow{\texttt{put}[\sigma_i]} P'_i\}. \end{aligned}$$

Proposition 3.6 *Expansion law for global parallel compositions*
Let $V \equiv (V_0|V_1|\ldots|V_n)\backslash\beta \;\; n \geq 1$ *then*

$$\begin{aligned} V \sim & \sum\{\alpha.(V_0|\ldots|V'_i|\ldots|V_n)\backslash\beta : V_i \xrightarrow{\alpha} V'_i, \alpha \neq \beta\} \\ & + \sum\{\tau_{Net}.(V_0|\ldots|V'_i|\ldots|V'_j|\ldots|V_n)\backslash\beta : V_i \xrightarrow{\alpha} V'_i, V_j \xrightarrow{\bar{\alpha}} V'_j, i \neq j\} \end{aligned}$$

Definition 3.7 A global term is *finite* if it contains only finite summations (`SUM`) and no recursions (`REC`). A global term P is *serial* if it contains no parallel composition (CCS | or data-parallel $\langle\ldots\rangle$), restriction or relabeling, and also the defining equation of any recursion in P contains no parallel composition, restriction or relabeling.

By expansion, every finite global term can be equated to a finite serial and global term.

Proposition 3.8 *Let* V *and* W *be serial finite and global terms.* $V \sim W$ *iff* $\mathcal{A}_1 \vdash V = W$

A last remark on the process algebra is that CCS is said to have *interleaving semantics* because $a|b \sim a.b + b.a$ which ignores the possibility of a and b occurring simultaneously. BSPA has the same semantics. In the following section we will present a cost model that considers simultaneity.

4 Cost model

Parallel programming researchers commonly refer to a parallel performance model as a *cost model* and we will use this terminology. Following BSP, we will define a cost model where execution time is related to the number of steps of local processes and the estimate of the time required for a global communication superstep. The result is a cost model which conforms to the operational semantics. Since processes are considered modulo bisimulation, it is natural to require that the cost is stable for this equivalence. The main

consequence of this is that cost is oblivious to the existence of an alternate copy of any process. As for the bisimulation law $P + P \sim P$, this is a reasonable notion of parallel execution *time*: additional copies of the same process do not add to the worst case time.

We have restricted our study to the CCS strong bisimulation relation $\sim$ and ignore the so-called *weak* bisimulation $\approx$. This is justified as follows in the context of a cost model. Weak bisimulation would lead (by $\tau.P \approx P$) to basic cost $\mathcal{C}(\tau) = \mathbf{1}$ (for example, in the semi-ring [3] $(\mathbb{N} \cup \{-\infty\}, max, +, -\infty, 0)$ $\mathcal{C}(\tau) = \mathbf{1} = 0$) which amounts to assigning the same cost to $\tau^{100000}.P$ and to $\tau.P$ while the first term represents a large number of internal synchronizations. That is not coherent with the use of cost as an estimate of execution time. Other motivations for dismissing weak bisimulation are given in [1] in the context of information flow analysis.

In the following subsections we recall which properties a semi-ring has to verify, we build a specific semi-ring $\mathcal{S}$ based on a simple and generic semi-ring S adapted to the parallel properties of BSPA, we expose the cost rules and then we define a way to finally obtain a suitable cost.

4.1 Definition of the semi-ring $\mathcal{S}$

4.1.1 Basics on semi-rings

An idempotent semi-ring $(\mathcal{S}, \oplus, \odot, \mathbf{0}, \mathbf{1})$ has to verify that $\oplus$ is associative, commutative, idempotent and has $\mathbf{0}$ as unit. And that $\odot$ is associative, is absorbed by $\mathbf{0}$, has $\mathbf{1}$ as unit and is distributive over $\oplus$.

This theory is generic: it is based on any possible idempotent semi-ring. The elements of $\mathcal{S}$ are called *s*calars. Here are examples of possible instantiation:

- $(\mathbb{N} \cup \{-\infty\}, max, +, -\infty, 0)$ which computes longest paths, that is to say worst-case time complexity.
- $(\mathbb{N} \cup \{+\infty\}, min, +, +\infty, 0)$ which computes the earliest dead-lock time.
- $(\mathbb{B}ool, \vee, \wedge, false, true)$ which computes reachability in a transition system.
- $(\mathbb{P}(\texttt{Act}), \cup, \texttt{concat}, \emptyset, \epsilon)$ which computes trace languages.
- $(\mathbb{N} \cup \{-\infty\}, max, \lambda a@i.b@j \mapsto max(i, j), \infty, a@0)$ which computes the maximal number of processors used by the process.

4.1.2 Definition of the semi-ring $\mathcal{S}$

In this section, let $\mathcal{S}$ be any semi-ring. We need to measure paths in the graphs [3] which are semantics for CCS processes: the so-called *labeled transition*

systems whose nodes are processes, edges transitions and edge labels are the events.

The property $x \oplus x = x$ that $\oplus$ is idempotent is required by the $P + P \sim P$ law of processes and our demand that bisimulation should respect costs.

As a process cost is associated with the set of execution paths, it has to represent the barriers. The cost of a BSP program is unknown until a superstep is completed. The cost can be either a vector which represents the accumulated time on each processor before the barrier. It can be a triple which contains the information of the cost of a barrier bracketed by two adjacent asynchronous phases: the scalar represents the cost of the barrier, and the two vectors represent the cost of asynchronous computation before/after the barrier. A special kind of semi-ring is needed to express this and we will define it in two steps: from $\mathcal{S}$ to $\mathcal{S}^p$ and then to $\mathbb{S}$.

First, we define the monoid $S^p_{MULT} = (\mathcal{S}^p \cup (\mathcal{S}^p \times \mathcal{S} \times \mathcal{S}^p), \odot, \mathbf{1})$ where $\mathbf{1} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$ and where $\odot$ applies to vectors point-to-point.

Definition 4.1 Let $v = \begin{pmatrix} v_0 \\ \vdots \\ v_{p-1} \end{pmatrix}$ then $\sum v = v_0 \oplus \ldots \oplus v_{p-1}$.

Definition 4.2 We define the operation $\odot$ on this monoid. v, w, v' and w' are vectors. x and x' are scalars.

$\odot$	v'	(v', x', w')
v	$v \odot v'$	$(v \odot v', x', w')$
(v, x, w)	$(v, x, w \odot v')$	$(v, x \odot \sum(w \odot v') \odot x', w')$

As two vectors are concatenated, they are simply concatenated point-to-point (abbreviated to $v \odot v'$). As a triple is followed by a vector, the vector is concatenated point-to-point to the last, or "future" vector of the triple. As a vector is followed by a triple, the vector is concatenated point-to-point to the triple's first vector. And when a triple is followed by a triple, that is to say between two barriers in our model, the two central vectors are concatenated and transformed into a scalar (with the $\sum$ operation), to be concatenated to the scalars of the triples.

Proposition 4.3 $\odot$ *is associative and has* $\mathbf{1}$ *as unit.*

From this associative monoid, we then build the semi-ring

$$\mathcal{S} = (\mathbb{P}_{\mathrm{FIN}}(\mathrm{S}^{\mathrm{p}}_{\mathrm{MULT}}), \cup, \odot, \mathbf{0}, \mathbf{1})$$

of finite sets of elements from the previous one, where $\mathbf{0} = \emptyset$ and $\mathbf{1} = \{\mathbf{1}\}$. Union on sets is associative, commutative and idempotent. For simplicity we will identify singleton sets with their unique elements as for $\mathbf{0}$ and $\mathbf{1}$ above.

The next step is to prove the distributivity of $\odot$ over $\oplus$ when $\odot$ is lifted to sets of values (vectors or triples), which is trivial, thanks to the $\cup$ properties.

4.2 Associating costs to processes

Our costs are evaluated on finite processes in serial form. We assume that restriction and parallel composition have been eliminated. $\mathcal{E}_{FIN}$ is the set of finite processes (with no recursion).

We will define $\mathcal{C} : \mathcal{E}_{FIN} \to \mathcal{S}$ in two steps:

(i) We eliminate $\backslash$ and $|$ from terms of $\mathcal{E}_{FIN}$ by applying the expansion law [9]. We also assign basic costs $\mathcal{C}(\alpha)$ to events α.

(ii) The remaining operators are then `let in`, ., + and 0 with variables. Property $\mathcal{C}$ is thus verified with Rules 4.2.1 where $P \rhd c$ means that $\mathcal{C}(P)$ is evaluated to semi-ring element c.

Given the variability of network traffic in grid applications, the BSP constants g, and L could here be considered as random variables.

By construction, this theory of costs is able to express both the interleaving (through $|$) and simultaneous (through $\langle \ldots \rangle_p$) occurrence of events.

4.2.1 Local rules

After having applied the expansion law, we only need the rules for the Prefix, Summation and Relabeling.

$$\mathcal{C}(0) = \mathbf{0}$$

$$\texttt{SUM}\frac{P \rhd c \quad Q \rhd c'}{P + Q \rhd c \oplus c'}$$

$$\texttt{ACT}\frac{P \rhd c}{\alpha.P \rhd \mathcal{C}(\alpha) \odot c}$$

$$\texttt{LET}\frac{Q[X \leftarrow P] \rhd c}{\texttt{let } X = P \texttt{ in } Q \rhd c}$$

4.2.2 Global rules

$\mathcal{C}$ is now extended to a function $\mathcal{C}$ from all finite serial terms to $\mathcal{S}$ [Section 4.1.2]. Finite serial and global terms V, obtained by the Expansion Laws 3.5, 3.6, contain neither $\langle \ldots \rangle_p$, nor $|$, but only ., +, 0 and `let in`. As the expansion laws are applied, occurrences of $T_{\sigma,\pi}$, $\langle \alpha @ i \rangle_p$ appear. $T_{\sigma,\pi}$ cost expresses the

BSP execution time for a communication and synchronization barrier. $\langle \alpha @ i \rangle_p$ costs commute when they are not co-located (see the different cases of $\mathbb{C}$ in Definition 4.4 below).

$$\mathbb{C}(0) = \mathbf{0} \ \text{(i.e.}\{\}\text{)} \qquad \texttt{SUM}\frac{V_i \rhd c \quad V_j \rhd c'}{V_i + V_j \rhd c \oplus c'}$$

$$\texttt{ACT}\frac{E \rhd c}{\alpha.E \rhd \mathbb{C}(\alpha) \odot c} \text{ where } \alpha \in \mathcal{L}_V \qquad \texttt{LET}\frac{U[X \leftarrow V] \rhd c}{\texttt{let } X = V \texttt{ in } U \rhd c}$$

Definition 4.4 $\mathbb{C}\colon \mathcal{E}_v \to \mathbb{S}$

(i) $\mathbb{C}(\langle \alpha @ i \rangle_p) = \vec{1}[i \mapsto C(\alpha)]$ i.e. the p-vector where $i \mapsto C(\alpha)$ and $j \mapsto \infty$ for $j \neq i$.

(ii) $\mathbb{C}(\alpha) = (\mathbf{1}, C(\alpha), \mathbf{1})$

(iii) $\mathbb{C}(T_{\sigma,\pi}) = (\mathbf{1}, (\sum h_i^+ \oplus \sum h_i^-)^g \odot L, \mathbf{1})$ (here $\mathbf{1}$ is the neutral element of the semi-ring $\mathcal{S}^p$), $h_i^+ = \sum_{j=0}^{p-1} |\sigma_{ij}|$ and $h_i^- = \sum_{k=0}^{p-1} |\sigma_{ki}|$ where $|\sigma|$ is the syntactic size of substitution σ and the exponent $\_^g$ denotes repeated $\odot$ product. If $\sigma = X_1 \leftarrow P_1, \ldots X_n \leftarrow P_n$ then $|\sigma| = \sum |P_i|$. Where $|P|$ is the syntactical size of P.

4.2.3 Costs respect bisimulation

Lemma 4.5 (Stability) *Stability for global finite serial terms V and W: $V \sim W \Rightarrow \mathbb{C}(V) = \mathbb{C}(W)$.*

4.3 Reducing cost to scalars: observation

We have built a vectorial BSP semi-ring $\mathbb{S}$ to represent faithfully the resource consumption of processes. This semi-ring has to represent especially interleaving. Even if $\mathcal{S}$ is totally ordered like for instance $(\mathbb{N} \cup \{-\infty\}, max, +, -\infty, 0)$, there is no obvious order between two computation costs in $\mathbb{S}$. We need an observer's point of view to obtain the usual cost notion: a scalar for a set of vectorial costs and a total order on costs. To this end we define $\mathcal{O}bs$ an observation function which transforms our element of $\mathbb{S}$ into a scalar of $\mathcal{S}$. And then, we have tools to compare the costs. However we will note that this observation is not compositional because it acts like a barrier: in general $\mathcal{O}bs$ does not preserve $\odot$.

Definition 4.6 We define $\mathcal{O}bs : \mathbb{S} \to \mathcal{S}$:

- $\mathcal{O}bs(v) = \sum v$ Vectors are flattened.
- $\mathcal{O}bs((v, x, w)) = \sum v \odot x \odot \sum w$ Triples are reduced: the two vectors are flattened and added to the central scalar.

- And this function is then lifted to sets (elements of $\mathfrak{S}$) as follows:
  - $\mathcal{O}bs(\{X\}) = \mathcal{O}bs(X)$ where X can either be v or (v, x, w)
  - $\mathcal{O}bs(A \cup B) = \mathcal{O}bs(A) \oplus \mathcal{O}bs(B)$.

4.4 Examples

In this section $\mathfrak{C}(\alpha) = a, \mathfrak{C}(\beta) = b, \ldots$. We use the semi-ring $(\mathbb{N} \cup \{-\infty\}, max, +, -\infty, 0)$ which computes a longest path in the transition system, i.e. worst-case execution time.

- ASYNC

$$\mathfrak{C}(< \alpha, \beta >) = \mathfrak{C}((\alpha@0.\beta@1) + (\beta@1.\alpha@0)) = \mathfrak{C}(\alpha@0.\beta@1) \oplus \mathfrak{C}(\beta@1.\alpha@0)$$
$$= \begin{pmatrix} a \\ b \end{pmatrix} \oplus \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} a \\ b \end{pmatrix}. \ \mathcal{O}bs(\begin{pmatrix} a \\ b \end{pmatrix}) = Max(a, b).$$

- ISOCHRONE with $p = 4$

  We develop here the example already seen in Figure 2 of Section 2.3. First we need to compute the π_i and their application to the P'_i. Then we need to compute the different h_i where $|\alpha| = |\beta| = |\gamma| = |\delta| = t$ and $a = b = c = d = w$.

  $Max\ h_i^+ = Max\ h_i^- = 3t$. $\mathfrak{C}(V) = \mathfrak{C}(T_{\sigma,\pi}) \odot \mathfrak{C}(x)$. $\mathfrak{C}(T_{\sigma,\pi}) = (\vec{1}, 3tg + L, \vec{1})$

$$\mathfrak{C}(s) = \begin{pmatrix} w \\ w \\ w \\ w \end{pmatrix} \quad \mathfrak{C}(V) = (\vec{1}, 3tg + L, \begin{pmatrix} w \\ w \\ w \\ w \end{pmatrix}) \quad \mathcal{O}bs(\mathfrak{C}(V)) = 3tg + L + w.$$

5 Application to meta-computing: scheduling

We now have a complete cost model. We can apply it to usual scheduling problems in grid computing. A typical situation is to schedule two data-parallel processes on the same parallel machine without interference while maximizing performance.

We outline how our model can describe this situation, leading to the possibility of choosing the most efficient scheduling solution while ensuring a coherent overlapping of the two data-parallel processes.

We will use $p = 2$, but the example is easily extensible to larger p. Assume two data-parallel applications P and Q each one possibly executed on one or two processors. The problem is to find the fastest coherent scheduling of the two processes on a two-processor system.

We use the semi-ring $(\mathbb{N} \cup \{-\infty\}, max, +, -\infty, 0)$. For these examples, we introduce a new notation: $\langle P, \ldots, P \rangle_p$ will be abbreviated $\langle P : i = 0, .., p-1 \rangle_p$. Let $P = \langle b.\texttt{put}[].b.\texttt{put}[] : i = 0, \ldots, 1 \rangle_p$, $Q = \langle a.\texttt{put}[\sigma].a.\texttt{put}[\sigma] : i = 0, \ldots, 1 \rangle_p$, and $\not\Gamma(a) = t_a$, $\not\Gamma(b) = t_b$. Let σ be a set of symmetric "heavy" substitutions, that is to say, substitutions that have a significant size. Then, $\not\Gamma(T_{\sigma,\pi}) = gh+L$, $\mathcal{O}bs(\not\Gamma(P)) = 2t_b+2L$ and $\mathcal{O}bs(\not\Gamma(Q)) = 2t_a+2L+2gh$. As the control skeleton of a parallel program, P would be considered a compute bound process while Q is a communication bound process.

We enumerate possible scheduling depending on how many processors are used by each process.

(i) P and Q are distributed over both processors. If we naively use $P_I = \langle P_0|Q_0, P_1|Q_1 \rangle_p = \langle b.\texttt{put}[].b.\texttt{put}[]|a.\texttt{put}[\sigma].a.\texttt{put}[\sigma] : i = 0, .., 1 \rangle_p$, then P_0 may realize a barrier with Q_1 thus leading to incoherent (unwanted) communication. As a `put` can be captured by any other `put` it is useful to introduce a function $[\_, \_]$ to schedule local sub-processes. Without loss of generality, we suppose that the variables in P and Q are disjoint. $[\_, \_] : Proc^2 \to Proc$ is defined as follows:

- $[A, 0] = A$ and $[0, B] = B$
- $[a.A, b.B] = a.[A, b.B] + b.[a.A, B]$ where $a, b \neq$ put
- $[A + A', B] = [A, B] + [A', B]$ and $[A, B + B'] = [A, B] + [A, B']$
- $[\texttt{put}.A, b.B] = b.[\texttt{put}.A, B]$ and $[a.A, \texttt{put}.B] = a.[A, \texttt{put}.B]$
- $[\texttt{put}[\sigma_0].A, \texttt{put}[\sigma_1].B] = \texttt{put}[\sigma_0].\texttt{put}[\sigma_1].[A, B]$

Then we can rewrite P_I in $P'_I = \langle [P_0, Q_0], [P_1, Q_1] \rangle_p$:

$P'_I = \langle \texttt{let}\ \ X = \texttt{put}.\texttt{put}[\sigma].0\ \ \texttt{in}$
$\texttt{let}\ \ Y = a.b.X + b.a.X\ \ \texttt{in}$
$\texttt{let}\ \ Z = \texttt{put}[].\texttt{put}[\sigma].Y\ \ \texttt{in}\ \ a.b.Z + b.a.Z : i = 0, .., 1 \rangle_p$

$\not\Gamma(P'_I) = (\begin{pmatrix} t_a + t_b \\ t_a + t_b \end{pmatrix}, t_a + t_b + 4gh + 4L, \mathbf{1})$. $\mathcal{O}bs(\not\Gamma(P'_I)) = 2t_a + 2t_b + 4gh + 4L$.

(ii) P is *serialized* on one processor $\langle P_{ser} = b.b.\tau.b.b.\tau \rangle_p$ and Q is distributed over both processors. $P_{II} = \langle P_{ser}|Q_0, Q_1 \rangle_p =$
$\langle b.b.\tau.b.b.\tau|a.\texttt{put}[\sigma].a.\texttt{put}[\sigma], a.\texttt{put}[\sigma].a.\texttt{put}[\sigma] \rangle_p$
$\mathcal{O}bs(\not\Gamma(P_{II})) = 2t_a + 4t_b + 4gh + 2L$

(iii) Same as the P_{II} where Q is *serialized* on one processor $Q_{ser} = a.a.c_\sigma.a.a.c_\sigma$ where c_σ is a local action realizing the same substitution (σ) as `put[σ]`. Its cost is ph instead of pgh (locally, the g factor costs 1). P is distributed over both processors.
$P_{III} = \langle P_0, P_1|Q_{ser} \rangle_p = \langle b.\texttt{put}[].b.\texttt{put}[], b.\texttt{put}[].b.\texttt{put}[]|a.a.c_\sigma.a.a.c_\sigma \rangle_p$

$$\mathcal{O}bs(\mathbb{C}(P_{III})) = 4t_a + 2t_b + 4h + 2L$$

(iv) Each process is *serialized* on one processor.

$$P_{IV} = \langle P_{ser}, Q_{ser} \rangle_p = \langle b.b.\tau.b.b.\tau, a.a.c_\sigma.a.a.c_\sigma \rangle_p$$
$$\mathcal{O}bs(\mathbb{C}(P_{IV})) = \max(4t_b, 4t_a + 4h).$$

From the above cost estimates it is possible to select the scheduling with least execution time. In general, it depends on the numerical values of g and L, but symbolic comparisons may be possible. For example, we know symbolically either $P_{IV} < P_{II}$ or $P_{IV} < P_{III}$ depending on the result of $\max(4t_b, 4t_a + 4h)$. Both cases suggest that it is useless to distribute only one of the two programs compared to serializing both of them. Here is the explanation for the first case (easily applicable to the second case): as $\mathbb{C}(P_{Ser}) > \mathbb{C}(Q_{Ser})$, $\mathbb{C}(Q_{Ser}) > \mathbb{C}(Q_i)$ and $\mathbb{C}(Q_0) = \mathbb{C}(Q_1)$, $\mathbb{C}(\langle P_{Ser}|Q_0, Q_1 \rangle_p) = \max(\mathbb{C}(P_{Ser}|Q_0), \mathbb{C}(Q_1)) = \mathbb{C}(P_{Ser}|Q_0) = \mathbb{C}(P_{Ser}) + \mathbb{C}(Q_0)$, hence $\mathbb{C}(P_{II}) > \mathbb{C}(P_{IV})$.

It is clear from this example that solutions to large-scale versions of this problem can be computed mechanically.

Another typical meta-computing problem is to distribute a process like $P = \langle P_0, \ldots, P_3 \rangle_p$ on two 2-processor systems, while preventing blocking and minimizing execution time. The resulting process is of the form $\langle P_0, P_1 \rangle_p | \langle P_2, P_3 \rangle_p$ and should realize a 4-process barrier by two 2-process barriers and some point-to-point synchronizations (rule `PARV`) between the two systems. A technical problem is to send messages from one system to the other via the point-to-point synchronization. We propose a simple solution in [8]: to introduce a "substitution passing" rule which generalizes both CCS value-passing and our BSP communications.

6 Conclusion

This study highlights the fact that the CCS "interleaving" semantics can be reconciled with the data-parallel (hence BSP) notion of simultaneous actions. Moreover, a generic cost model can be designed to generalize both a standard notion of paths in CCS transition systems and the BSP notions of simultaneous parallel execution with communication and synchronization barriers.

Krishnan's distributed CCS model [5] and Rebeuf's asynchronous performance model [12] allow arbitrarily complex communications without taking advantage of barriers which prevent combinatorial explosion either in process algebraic calculations or in cost calculations. We have therefore favored the BSP point of view on parallel computation.

[4] propose time models. Time on transitions is given in term of probabilities and preserves the expansion law. This model does not consider the composition of actions. Our model does and proposes adapted algorithms for

a global calculus. We could immerse probabilistic models in our schema but we would need to compute the composition of two path by the sum (the product in $(\mathbb{N} \cup \{-\infty\}, max, +, -\infty, 0)$) of two random variables: the time of the first transition + the time of the following transition. The use of this model would depend on the complexity of this computation, which we did not study yet. Thanks to the algebraic properties of our model, what we do manually in Section 5 could be done mechanically by classical algorithms on matrices [3].

Our model's application to large-scale studies will require: a symbolic treatment of the width of data-parallel vectors and of individual machine identifiers, as well as the design of unification or constraint-solving algorithms. We will then be able to apply our formalism to the design of correct, safe and optimal scalable meta-computing processes, in particular through automatic analysis tools like model-checkers. Other extensions we are considering are mobility ([11,2]) and its application to security properties ([1]) in the context of parallel systems. This last problem is the object of a current project between LIFO and CEA where the formalism developed here will give a realistic model of data-parallel distribution and performance.

Another application of our theory we are investigating concerns denial of service attacks. In [6], Lafrance and Mullins model the memory cost of transactions from the point of view of a server. Our model refines their analysis by allowing the memory cost to increase and *decrease* along traces. Another improvement would be to have a binary constructor for $\langle \ldots \rangle$.

References

[1] S. Anantharaman and G. Hains. A synchronous bisimulation-based approach for information flow analysis. In *Third Workshop on Automated Verification of Critical Systems: (AVOCS'03)*, April 2003.

[2] G. Fournet and G. Gonthier. The reflexive CHAM and the join-calculus. In *23rd annual ACM Symposium on Principles of Programming Languages (POPL'96)*, pages 371–385, St.Petersburg (FLA) USA, January 1996.

[3] M. Gondran and M. Minoux. *Graphes et Algorithmes*, chapter 3. Eyrolles, 1985. Les Algèbres de Chemins.

[4] Joost-Pieter Katoen and Pedro R. D'Argenio. General distributions in process algebra. *Lecture Notes in Computer Science*, 2090:375–430, 2001.

[5] P. Krishnan. Distributed CCS. In *Theories of Concurrency: Unification and Extension: CONCUR-91, LNCS:527*, pages 393–407. Springer, August 1991.

[6] S. Lafrance and J. Mullins. Using admissible interference to detect denial of service vulnerabilities. In *Sixth International Workshop in Formal Methods. Electronic Workshops in Computing (eWiC) by British Computer Society (BCS)*. In J.M. Morris, B. Aziz, and F. Oehl, editors, 2003.

[7] F. Loulergue, G. Hains, and C. Foisy. A calculus of functional BSP programs. *Science of Computer Programming*, 37(1–3):253–277, May 2000.

[8] A. Merlin and G. Hains. A bulk-synchronous parallel process algebra. Rapport de Recherche RR-2004-06, LIFO, Université d'Orléans, 2004. `http://www.univ-orleans.fr/SCIENCES/LIFO`.

[9] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[10] R. Milner. Operational and algebraic semantics of concurrent processes. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science*. North-Holland, MIT-Press, 1990.

[11] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I and II. *Information and Computation*, 100(1):1–40 and 41–77, September 1992.

[12] X. Rebeuf. *Un modèle de coût symbolique pour les programmes parallèles asynchrones à dépendances structurées*. Thèse de Doctorat d'Université, Université d'Orléans, décembre 2000.

[13] L. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103, August 1990.